

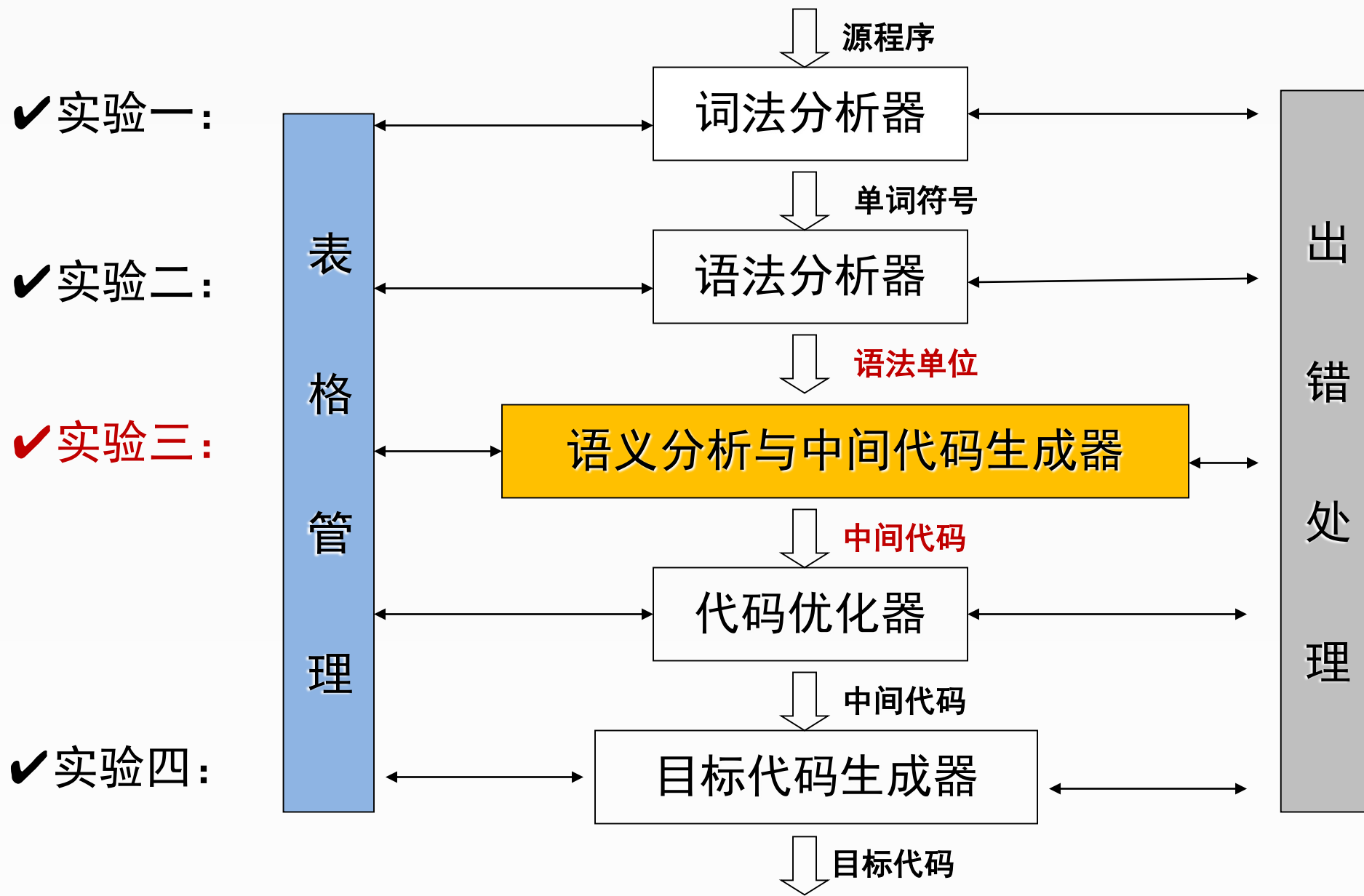


编译原理

实验三：典型语句的语义分析及中间代码生成

规格严格，功夫到家

编译程序的总体结构



实验目的

1. 巩固对**语义分析**的基本功能和原理的认识；
2. 加深对**自底下上语法制导翻译**技术的理解，掌握声明语句、赋值语句和算术运算语句的翻译方法。
3. 理解**中间代码**的表示形式，掌握三地址码的实现方式。

实验学时数：**4学时**

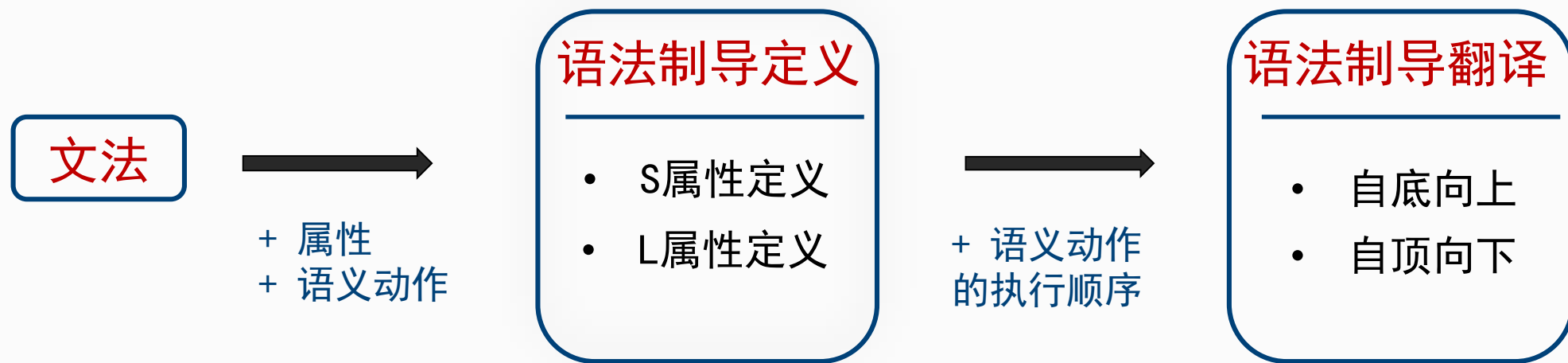
实验内容

1. 采用实验二中的文法，为语法正确的单词串设计翻译方案，**完成语法制导翻译**。
2. 利用该翻译方案，对所给程序段进行分析，输出生成的**中间代码序列**和更新后的**符号表**，并保存在相应文件中，中间代码使用三地址码的**四元式表示**。

注：更新后的符号表只需要保存id和type两个属性。

3. 实现**声明语句、简单赋值语句和算术表达式**的语义分析与中间代码生成（参考教材P260声明语句的翻译, P267赋值语句的翻译）。
4. 使用框架中的模拟器IREmulator验证生成的中间代码的正确性。

实验原理



使用语法制导翻译方案将语义检查和中间代码生成结合到语法分析中进行

S-属性定义的自底向上翻译

S-属性定义是只含有综合属性的语法制导定义，可以在自底向上的语法分析过程中被高效的实现。

注：综合属性指的是只能通过该节点自己或者子节点的属性来计算的属性

S-属性定义的翻译方案，只需要将每个语义规则放到相应产生式的末尾即可得到翻译模式，在对产生式规约时执行相应的语义动作。

S-属性定义的自底向上翻译

声明语句和简单赋值语句的翻译方案：

输入：

int a;

a = 3;

$S \rightarrow D\ id$

$D \rightarrow int$

$S \rightarrow id = E$

$E \rightarrow A$

$A \rightarrow B$

$B \rightarrow IntConst$

{p = lookup(id.name); if p != nil then enter(id.name, D.type) else error}

{D.type = int;}

{gencode(id.val = E.val);}

{E.val = A.val;}

{A.val = B.val;}

{B.val = IntConst.lexval;}

Lookup (name) : 查询符号表,
返回name对应的记录

gencode (code) : 生成
三地址指令code

enter : 将变量的类型填入符
号表

举例：语义分析栈

源语言：

Int *a*;

a = 4;

return (*a* + 8) * *a*;

中间代码：

(MOV, *a*, 4)

(ADD, \$1, *a*, 8)

(MUL, \$2, \$1, *a*)

(RET, \$2)

语法分析产生式列表：

D → *int*

S → *D id*

B → *IntConst*

A → *B*

E → *A*

S → *id = E*

B → *id*

A → *B*

E → *A*

B → *IntConst*

A → *B*

E → *E + A*

B → (*E*)

A → *B*

B → *id*

A → *A * B*

E → *A*

S → return *E*

语义分析栈执行动作说明：

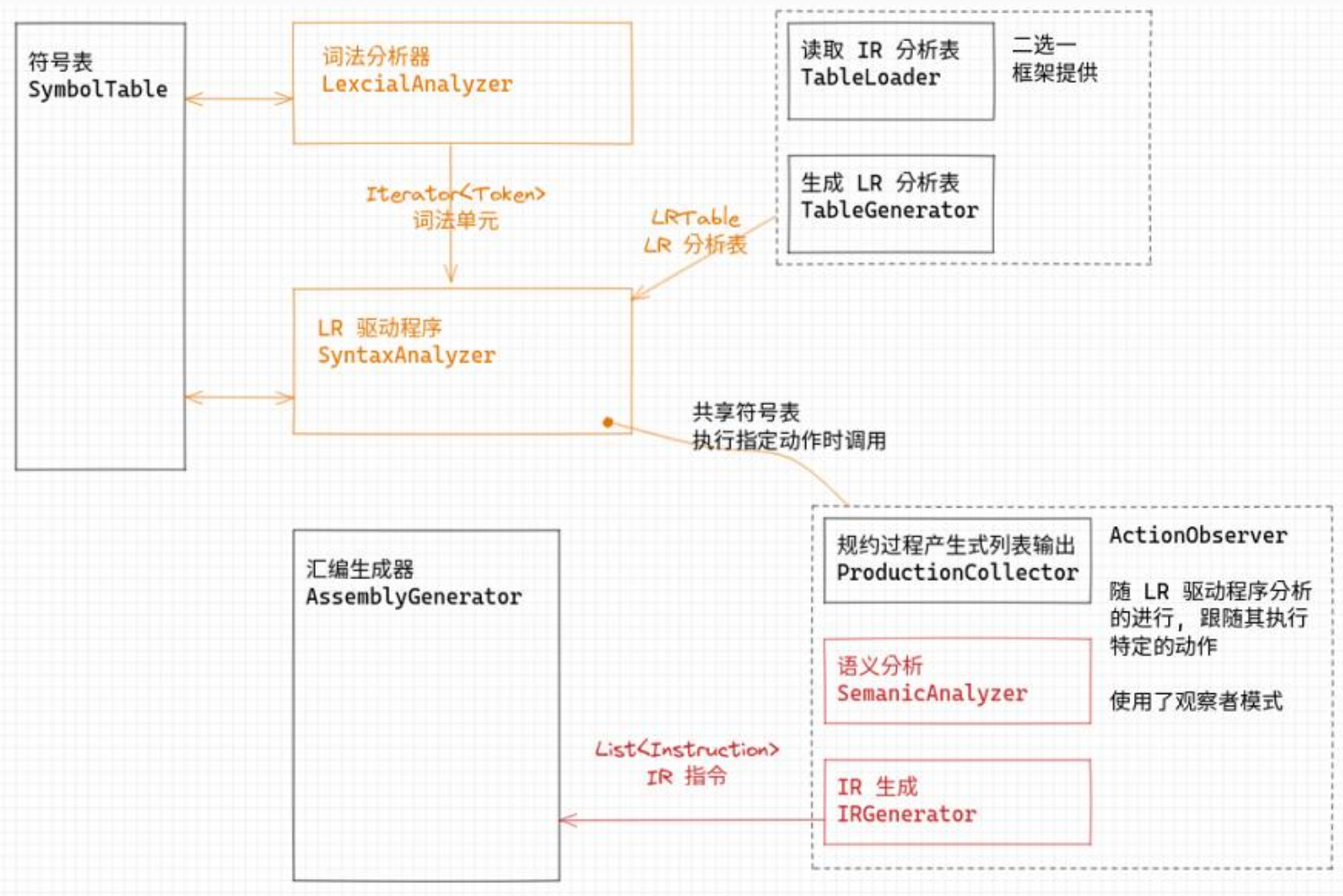
1、语义分析的结果是更新符号表，根据翻译方案，语义分析栈需要保存type属性；

2、自底向上的分析过程中Shift时将符号的type属性（从Token中获得）入栈；

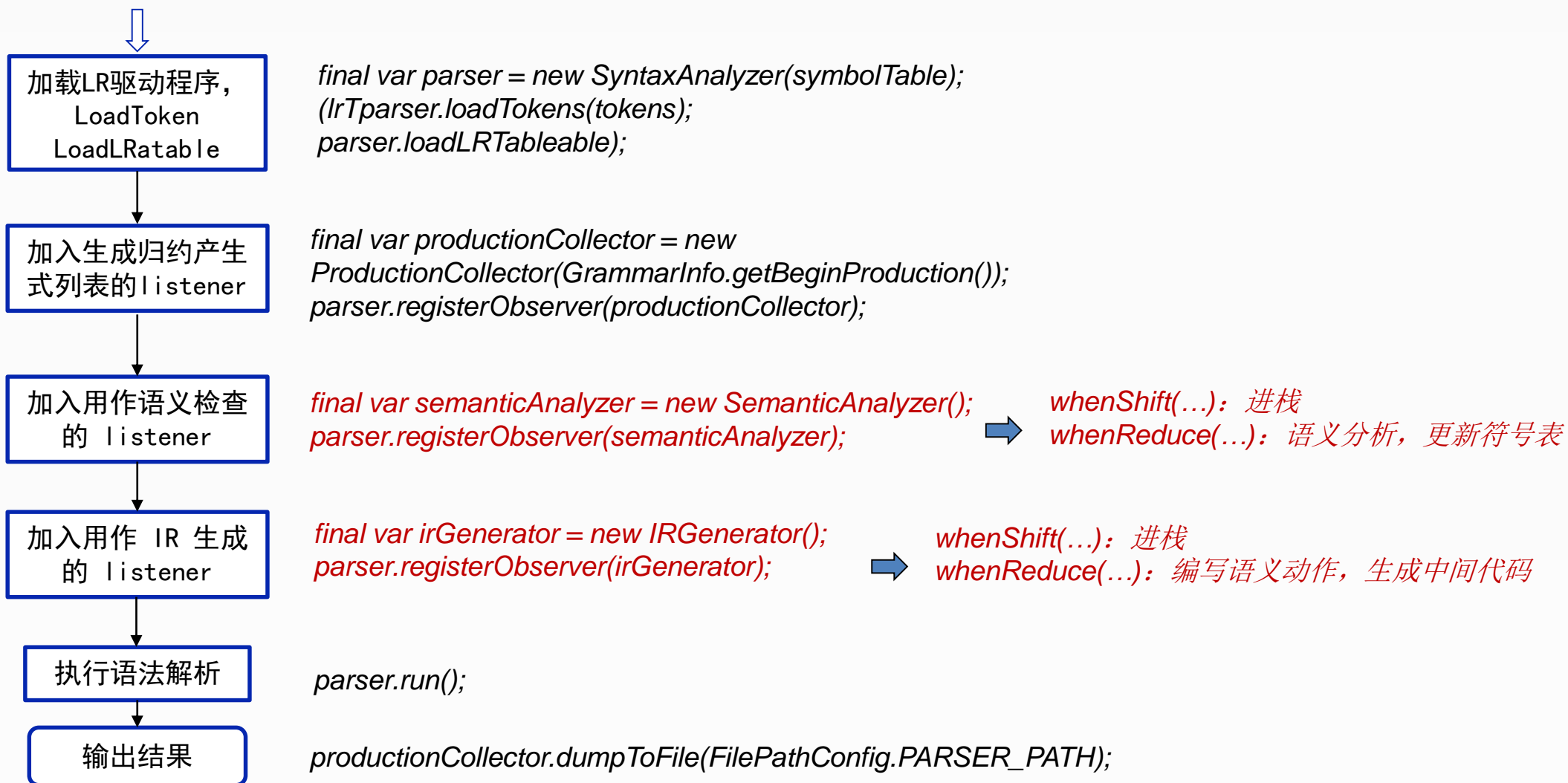
3、当*D*→*int*这条产生式归约时，*int*这个token应该在语义分析栈中，把这个token出栈，把它的type类型赋值给*D*的type后入栈；

4、当*S*→*D id*这条产生式归约时，取出*D*的type，这个type就是*id*的type，更新符号表中相应变量的type信息，压入空记录占位（*D id*被归约为*S*，*S*不需要携带信息）；

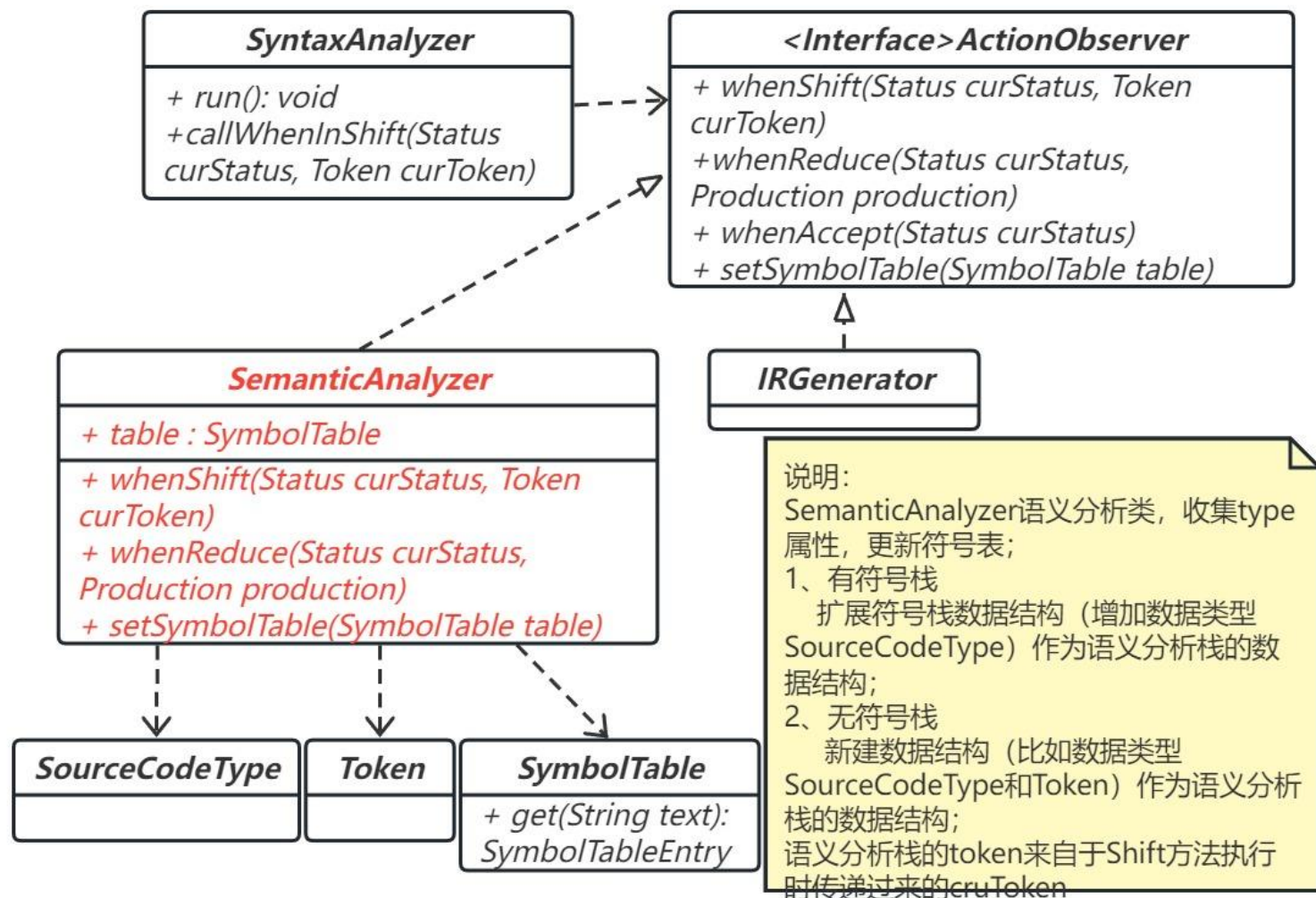
编译器程序整体框架图



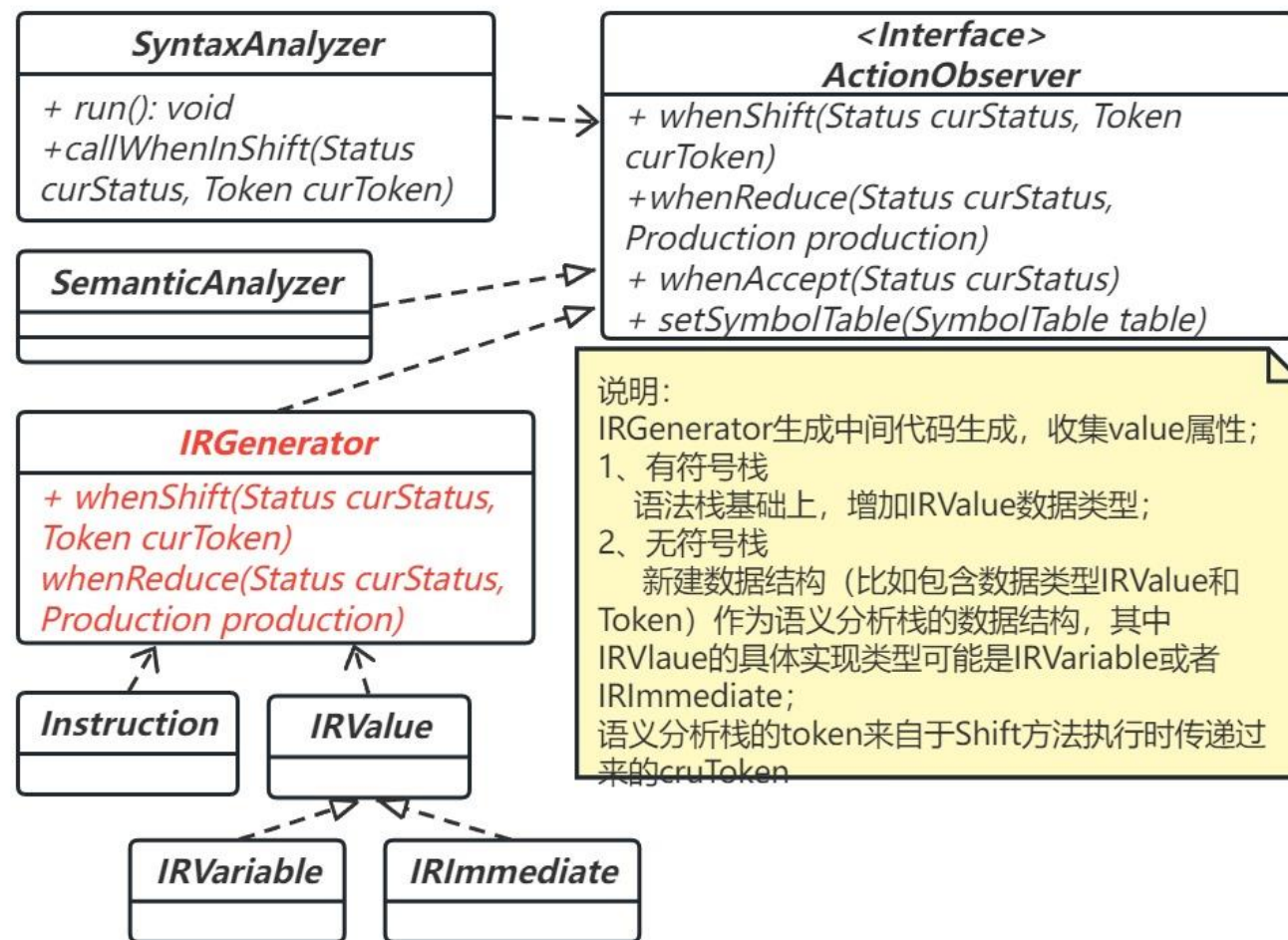
主程序流程说明



语义分析SemanticAnalyzer：在自底向上的分析过程中更新符号表。



中间代码生成IRGenerator：生成中间代码



实验步骤

- 1、设计文法的翻译方案；
- 2、设计各个观察者需要的数据结构；
- 3、实现各观察者在分析过程中的动作处理。

同学们，
独立开始实验